CPSC 2150 A5 Complexities

Jay Seung Yeon Lee
100 357 736

Part 1)

Quick Sort	1000 elements	2000 elements	4000 elements	8000 elements
Ascending	0.00154232	0.00621378	0.03000497	0.10653213
Descending	0.00173810	0.00584777	0.02878312	0.09383432
Random	0.00015154	0.00033226	0.00072393	0.00155797
Merge Sort	1000 elements	2000 elements	4000 elements	8000 elements
Ascending	0.00109708	0.00148366	0.00187657	0.00876652
Descending	0.00089949	0.00194371	0.00246137	0.00373634
Random	0.00087003	0.00204390	0.00187154	0.00505281
Insert Sort	1000 elements	2000 elements	4000 elements	8000 elements
Ascending	0.00191570	0.00848586	0.03840201	0.15189011
Descending	0.00000001	0.00259151	0.01427586	0.13615130
Random	0.00172705	0.00846926	0.03718412	0.15637959

Part 2)

Note all complexity calculations were done with 4000 elements and 8000 elements to minimize potential error

Cases for Quick Sort:

1. Best Case: Random Order
2. Avg Case: Ascending Order
3. Worst Case: Descending Order

Best case and average case for Quick Sort is O(n log (n)), however it's worst case may perform at O(n^2).

For best case of Quick sort: random order, 4000 elements time 72393 / 4000 is 18. In comparison, 8000 elements time is 155797 / 8000 is 20.

From this we can assure time complexity increased per increase in number of element is very minimal and is less than linear. For random insertion, complexity is closer to the base case (O(n log(n)).

For worst case: Ascending Order, 4000 elements time is 3000497 / 4000 which is 750 and 8000 elements 10653213/ 8000 which is 1331. From the time complexity calculation, we can derive that time complexity for ascending order is close to being a linear.

Cases for Merge Sort:

1. Best Case: Ascending Order
2. Avg Case: Random order
3. Worst Case: Descending Order

Merge Sort has a best, average, and worst case of O(n log(n))

Best case for merge sort is descending order. With 4000 elements resulting in 246137/ 4000 = 61 and 373634 / 8000 which is 46.7. I've noticed that the discrepancy between the expected result and actual result is due to the cost of merge.

Cases for Insert Sort:

1. Best Case: Random Order
2. Avg Case: Ascending Order
3. Worst Case: Descending Order

Insertion sort has an average and worse case of O(n^2) and best case of Linear O(n) complexity.

Worst case of insertion sort is Descending order which is 4000 elements 1427586/ 4000 = 356 and 8000 elements 13615130/ 8000 which is 1701, closer to average and worst case of O(n^2).

Best case is Ascending order which is 3718412 / 4000 = 929.603 and 15637959/ 8000 = 1954 which is linear. (Random Order had similar result)

Part 3.

Quick Sort

Best case and why:

Random quicksort had the best efficiency of the three arrays because there is a very low probability of bad splits in random arrays.

Worst case and why:

Both ascending and descending are worst case for quick sort. Reason for this is because the picked pivot will always be the smallest or largest element of the array thus not using quicksort efficiently.

Merge Sort

Best case and why:

Best case for merge sort is when elements are already in sorted ascending order. If all of the elements of the first array are less than elements in the second array, the number of comparisons will be the same number as the number of the element, thus is the most efficient.

Worst case and why:

Worst case for merge sort is if the elements are sorted in a descending order. Since in this case, the minimum number of comparisons will be 2n.

Insert Sort

Best case and why:

Best case for insertion sort is if the element in the array is already sorted (meaning it is already in an ascending order). In this case, O(n) will be the complexity since the program will only have to make n number of comparisons in a list with n number of elements. For my case, Random ordered list had a similar result, but this may be due to the list being close to already-sorted state.

Worst case and why:

Descending Order will be the worst case as program will have to make n^2 comparisons as the entire list is at the very opposite of where they are supposed to be.